Enc. (1)

(12)

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered) UNCLASSIFIED

AD-A166 663

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) BROADCASTING TOPOLOGY AND ROUTING INFORMATION IN COMPUTER NETWORKS		5. TYPE OF REPORT & PERIOD COVERED paper
		6. PERFORMING ORG. REPORT NUMBER LIDS-P-1543
7. AUTHOR(s) John M. Spinelli		8. CONTRACT OR GRANT NUMBER(s) DARPA Order No. 3045/2-2-84 Amendment #11 ONR/N00014-84-K-0357
9. PERFORMING ORGANIZATION NAME AND ADDRESS Massachusetts Institute of Technology Laboratory for Information and Decision Systems Cambridge, Massachusetts 02139		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Program Code No. 5T10 ONR Identifying No. 049-383
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Boulevard Arlington, Virginia 22209		12. REPORT DATE March 1986
		13. NUMBER OF PAGES 13
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Office of Naval Research Information Systems Program Code 437 Arlington, Virginia 22217		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release: distribution unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

An efficient algorithm is presented which allows each node in a computer network to maintain a correct view of the network topology despite link and node failures. Reliability is achieved without transmitting any information other than the operational status of links. Messages are only sent in response to topological changes: periodic retransmission is not required.

The algorithm is extended to allow nodes to maintain congestion measurements used in making routing decisions.

86 4 15 003

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

March 1986 LIDS-P-1543

Broadcasting Topology and Routing Information in Computer Networks

John M. Spinelli[*]

Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

(Submitted to *IEEE Transactions on Communications*)

Abstract

An efficient algorithm is presented which allows each node in a computer network to maintain a correct view of the network topology despite link and node failures. Reliability is achieved without transmitting any information other than the operational status of links. Messages are only sent in response to topological changes: periodic retransmission is not required.

The algorithm is extended to allow nodes to maintain congestion measurements used in making routing decisions.

1 Introduction

At any time while a store and forward computer network is operating one or more of its communication links or processing nodes may malfunction or be put back into service. The recovery of the network from such a change in its topology is an essential part of providing reliable data communication. This paper is concerned with the problem of keeping each network node informed of the entire network topology when it is subject to occasional changes over time. Any network which

[*]This research was conducted at the MIT Laboratory for Information and Decision Systems and was supported in part by a National Science Foundation Graduate Fellowship, by the National Science Foundation under Grant NSF-ECS-8310698, and by the Defense Advanced Research Projects Agency under Grant ONR/N00014-84-K-0357.

uses decentralized adaptive routing needs to address this problem. The classic example of this is the ARPANET, where each node maintains a map of the entire network and uses it in making routing decisions [1,2]. Even networks which use some form of hierarchical routing need to solve this problem within some level of the hierarchy.

Topological changes may occur at any time. Since all messages sent in the network are subject to delay, a node can never be certain that it knows the correct topology at some instant of time. However, distributed algorithms can be designed to guarantee that each node is made aware of the correct status of each link to which it is connected[1], provided that the topology does not change for a sufficient but finite time. Algorithms which accomplish this are called topology algorithms.

A topology algorithm is a set of rules governing the topology information stored at a node, as well as the contents, transmission, and reception of algorithm messages. These messages are called *topology updates* or *update messages*. They usually contain an indication of the operational status (up or down) of one or more network links. When a network is started or reinitialized the algorithm must determine the initial network topology and communicate it to every node. Thereafter, the topology information must be kept current by update messages which are sent periodically or in response to a topological change.

Although it might appear that any effective method of broadcast could serve as a topology algorithm, there are several subtle issues which cause difficulties.

1. The links which are used to communicate topology information are themselves subject to fail at any time. The failure of a link creates additional topology information which must be communicated. It may also block the transmission of previous topology information about other links, and in the worst case may divide the network into two disconnected sets of nodes.

2. A link may experience several topology changes within a short time. Other network nodes must be able to determine which change was the most recent. In general, nodes must be able to distinguish between old and new information about the status of a link.

3. While a topology algorithm is running, additional topology changes may occur. The topology algorithm must be capable of either incorporating new information during execution, or of starting a new algorithm version. If different versions are used, each node must be able to determine which is the most recent version, a problem similar to 2 above.

[1]A node is said to be connected to a link if there is a path of operating links from the node to either end node of the link.

4. The repair of a single link can cause two parts of the network which were disconnected to reconnect. Each part may have arbitrarily out-of-date topology information about the other. The algorithm must ensure that the two parts eventually agree, and adopt the correct network topology.

In order to overcome these difficulties, algorithms have been designed which are quite complex [3,4,5,6,7]. A common approach is to include auxiliary information such as message counters, sequence numbers, or age fields in update messages along with the topology information itself [2,4,5]. The sequence numbers are used to distinguish between old and new messages. Some algorithms employ several types of messages for different purposes such as initialization, acknowledgements, and termination. Update messages can be sent periodically, or in the case of event driven algorithms, only in response to topological changes. Algorithms may respond to a topological change by rebuilding the entire network topology, or by modifying an existing topology to reflect the change. Lastly, some algorithms provide theoretical guarantees of correctness, while others are designed to work correctly in all but the most unusual circumstances.

In this paper we take a rather unconventional approach to solving the topology problem. The algorithm which we present, called the *Shortest Path Topology Algorithm* (SPTA), uses no auxiliary information at all. The update messages consist only of topological information, i.e. link status information. The algorithm is purely event driven in that nodes transmit messages only in response to receiving a topology update message from a neighbor, or to detecting a status change in an adjacent link. It does not rely on periodic retransmission of messages, or use timers or clocks of any kind. The update messages are used by a node to modify its existing topology. There is only one type of message used, and no special cases for reconnection of disconnected parts of the network. The simple message structure allows a theoretical proof of correctness that is rather straight forward.

There are two main motivations for constructing an algorithm with the above characteristics. Although the use of auxiliary information is a logical way to deal with the four difficulties mentioned earlier, it also introduces additional problems [8]. For example, if sequence numbers are used, the finite bit field used to store them may eventually wrap around. While this can be avoided by choosing a large bit field, some provisions should still be made for resetting the numbers. The introduction of auxiliary information into update messages usually leads to increasing complexity. An algorithm which avoids auxiliary information entirely avoids also the complexities and difficulties associated with it. Apart form this there is the academic question of whether topological information alone is sufficient to solve the topology problem. SPTA shows that although auxiliary information may be desirable in some circumstances, it is not required in topology algorithms. By exchanging only link status information, the network nodes can arrive at and maintain a correct topology.

2 The SPTA Algorithm

The main idea behind SPTA is that each node is viewed as trying to construct the network topology based on reliable information that it has about the status of adjacent links, and possibly unreliable information (in the sense that it might be outdated) that it has received from neighboring nodes. Each piece of link status information that a node receives from a neighbor is assigned a *distance measurement* based on the shortest path distance from the neighbor to the link in question. When trying to determine the status of a link, a node chooses the piece of information with the smallest distance measurement which it has received concerning that link. High reliability of link status information is associated with a small distance measurement.

2.1 Assumptions

The correct operation of any topology algorithm depends strongly on the way in which link status changes are detected by the network nodes. The status change of a node (node failure or repair) will be represented by a status change in each of the links adjacent to the node. When a link changes status this is detected by both end nodes, perhaps nonsimultaneously. Each end node decides whether a link is *up* (operating) or *down* (not operating). If one end node declares a link to be down, then the other end node must also declare it to be down before either can bring the link up. When a link is down, update messages are not received on it. When a link is up it provides error free communication with finite delay between its two end nodes, and maintains the order of transmitted messages. The foregoing assumptions merely imply that there is a reasonable data link control protocol operating between the two end nodes of a link, but leave a great deal of flexibility in the design of the protocol. Lastly, we assume that nodes maintain the integrity of data and messages which are stored in their memories.

2.2 SPTA Data Structures and Rules

Each node i in the network maintains a topology table T^i called its main topology table. A topology table is a list of the operational status of each link in the network. T^i contains and entry $T^i(l)$ for each link l, and reflects node i's current best estimate of the network topology. It is the official topology that would be used by a routing

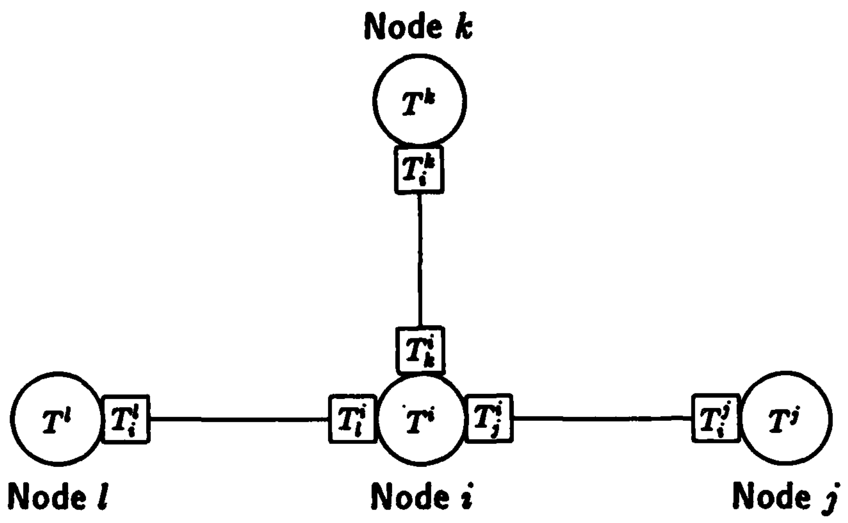

Figure 1: The Main and Port Topology Tables for a Simple Network.

algorithm operating at node i. In addition to its main topology table, node i maintains a port topology table T_j^i associated with each neighboring node j. The entry in this table for link l is denoted $T_j^i(l)$. The superscript indicates that the table is stored at node i, while the subscript indicates that it is associated with neighboring node j. The information stored in table T_j^i is the latest topology information received by node i from node j. The tables stored at node i are shown in Figure 1. When a node's main topology table changes, it sends a message notifying each of its neighbors of the change. Therefore, T_j^i is essentially a delayed version of node j's main topology table, T^j. The only exception to this is for links which are adjacent to node i. Since i always knows the status of its adjacent links, it can enter this information directly into each of its port tables and its main table. Any update message which i receives concerning its adjacent links is ignored.

The following communication rules are used to maintain the topology tables described above.

1. When an entry in a node's main topology table changes, a message containing the new value is sent on each operating adjacent link.

2. When a node receives a topology change message from a neighbor which concerns a nonadjacent link, it enters the change in the port topology table associated with that neighbor.

3. When a node detects that an adjacent link has become operational, it transmits its main topology table over that link.

Rule 3 is needed since while a link is down, the port topology tables associated with it can become arbitrarily out-of-date. Therefore, when the link becomes operational the entire port table needs to be refreshed.

The remaining portion of SPTA deals with how a node constructs its main topology table based on the contents of its port topology tables. Since each topology table specifies a graph, we can perform a shortest path operation using a table and considering each link to have a distance of 1. Let *shortest-path*(j, l, T^i_j) be the length (in links) of the shortest path from node j to the closest end node of link l according to the port topology table T^i_j. Using this, we define the following distance measurement.

$$D^i_j(l) = 1 + \textit{shortest-path}(j, l, T^i_j)$$

For links l which are disconnected from node j according to T^i_j, $D^i_j(l)$ is considered to be arbitrarily large. High reliability of link status information $T^i_j(l)$ is associated with a small value of $D^i_j(l)$. When link (i, j) is down, node i calls port table T^i_j *inactive*, and disregards all information stored in it. An inactive port table becomes *active* after an update message is received from the neighbor associated with that table. This insures that the table has been refreshed via rule 3 before it is used.[2]

Node i maintains its main topology table according to the following rules.

4. When an adjacent link changes status, the change is entered in each of node i's port topology tables and its main table.

5. When an entry in one of node i's port topology tables T^i_j changes, it recalculates $D^i_j(l)$ for each nonadjacent link l. It then constructs its main topology table by setting $T^i(l)$ equal to $T^i_{j'}(l)$, where $T^i_{j'}$ is the *active* port table with the smallest distance measurement for link l. If l is disconnected according to each active port table, then $T^i(l)$ is not modified.

In rule 5 above a node is in effect polling its port tables for each nonadjacent link l and choosing the entry with the smallest distance measurement. Ties can be broken arbitrarily or according to a fixed ordering of port tables. Together, rules 1 through 5 completely specify SPTA.

2.3 SPTA Implementation Outline

The following is an outline for an implementation of SPTA which refers to the above rules. Consider the algorithm operating at each node i. An active neighbor of i is

[2] Waiting to receive a message before calling a port table active is not required for correct operation, but avoids temporarily propagating out-of-date information.

one connected by an operating link.

A. (A node processor is reinitialized.)

1. Each adjacent link is initially considered down.
2. All entries in the main and port topology tables are set to down.

B. (A message is received from neighboring node j concerning the status of link l.)

1. The new status of l is entered in $T_j^i(l)$.
2. $D_j^i(k)$ is recomputed for each link k.
3. The main topology table is reconstructed.
4. Any changes in T^i are sent to all active neighbors.

C. (Adjacent link l goes down)

1. The new status of l is entered in each port table and the main table.
2. $D_j^i(k)$ is recomputed for each link k and each neighbor j.

3 & 4. Same as B.3 and B.4.

D. (Adjacent link l goes up.)

Same as in C except that the entire new main topology table is sent on link l.

2.4 Correctness Proof of SPTA

Assume that at some time t_s a network is in steady state. This means that for each node within a connected component of the network, the main topology tables are correct for each link which is adjacent to a member of the component. In addition, no algorithm messages are being transmitted. Note that a node is in steady state immediately after being reinitialized. Between time t_s and some later time t_0 an arbitrary number of link topology changes occur. For a sufficient but finite time interval after t_0 no further topology changes occur.[3] We wish to show that at some later time $t_f \geq t_0$ steady state has been reestablished. Let T^* be the correct network topology that an omniscient observer would see upon examining the network after t_0. We say that node i "knows the correct topology" if its main topology T^i agrees

[3]This excludes situations where the topology is continuously changing. SPTA is not applicable in these situations.

with T^* for all links that are connected to i. We begin by showing the following theorem.

Theorem 1. *SPTA works correctly in the sense that, under the preceding assumptions, there is a finite time t_f after which each node knows the correct topology.*

Proof. In what follows, we say that a link l is at distance n away from i if in the graph defined by T^* the shortest path from i to the closest end node of l is n hops long. We will show by induction that for each integer $n \geq 0$ there is a time $t_n \geq t_s$ after which, for each node i, T^i agrees with T^* for all links that are at a distance of n or less from i. The induction hypothesis is clearly true for $n = 0$ since each node i knows the correct status of its adjacent links and records them in its main topology table T^i. We first establish the following lemma.

Lemma 1. *Assume that the induction hypothesis is true for time t_n. Then there is a time $t_{n+1} \geq t_n$ after which the port topology table T^i_j, for each active neighbor j of node i, agrees with T^* for each link at a distance n or less from j.*

Proof. Consider waiting a sufficient time after t_n for all messages which were sent from j to i before t_n to arrive. By rules 1, 2, and 3 of the algorithm, T^i_j agrees with T^j for all links which are not adjacent to i. Therefore, by the induction hypothesis T^i_j agrees with T^* for each link at a distance of n or less from j which is not adjacent to i. By rule 4 the correct status of links adjacent to i are recorded in T^i_j. Therefore, T^i_j also agrees with T^* for all links adjacent to i. This proves the lemma.

To complete the proof of Theorem 1 we must show that after time t_{n+1}, T^i agrees with T^* for all links l which are at a distance $n+1$ from i. Each neighboring node of i must be at a distance of at least n from l. Let J be the nonempty subset of i's neighbors which are at a distance of exactly n from l. Consider a port topology table T^i_j such that $j \in J$. By lemma 1, T^i_j agrees with T^* for all links at a distance of n or less from j. From rule 5 of the algorithm, i will calculate a distance measurement for link l of $D^i_j(l) = n + 1$. This is true for each neighbor $j \in J$. Now consider each neighbor $j \notin J$. By rule 5 of the algorithm, node i will calculate $D^i_j(l) > n + 1$. If this were not true, it would imply that in the graph described by T^i_j there is a working path of n links or less connecting j to one of the end nodes of l. But by lemma 1, T^* and T^i_j agree on the status of these links and no such path exists. Therefore, by rule 5 $T^i(l)$ will agree with $T^i_j(l)$ for $j \in J$. This implies that T^i will agree with T^* about link l and completes the proof of Theorem 1.

After t_f a node i's main topology table will not change concerning links to which i is connected. Rule 5 states that T^i will not change concerning disconnected links either. Since a node only transmits messages when its main topology changes, node i will not transmit any messages after t_f. A finite time later a condition of steady state will be reestablished. This completes the correctness proof of SPTA.

3 Algorithm Characteristics

The time complexity [9] of SPTA can be established by examining its correctness proof. We are interested in the time from t_s until the algorithm terminates, assuming that each message takes one time unit to transmit. At time t_d (where d is the network diameter) Theorem 1 is satisfied, and one time unit later the algorithm terminates. Therefore, for an n node network the time complexity is $O(n)$.

The communication complexity for single link topology changes can be established by examining rule 1 of SPTA. When a node's main topology table changes, it sends a message on each of its adjacent links. This results in $2l$ messages being sent on an l link network and gives $O(l)$ communication complexity. Unfortunately, when multiple link failures occur, there are unusual situations in which this result does not hold. Figure 2 illustrates one such situation. When node 1 detects that link B has failed, it considers port table T_2^1 inactive and ignores all information contained in it. To determine the status of link A node 1 looks to its other port table T_3^1. Since the message "$(3 \rightarrow 1,\ A\downarrow)$" sent in event 2 has not yet arrived, T_3^1 still lists link A as up. Node 1 temporarily has an incorrect status for link A and sends it to node 3. The situation is corrected as soon as the above delayed message arrives in event 4. Notice that node 1 sends three messages to node 3 concerning link A, whereas A's status has actually changed only once. If such examples are carried to their extreme, it can be shown that the maximum number of messages sent increases as 2^k, where k is the number of link status changes [10]. Although this behavior is undesirable, it can only occur in rare situations such as when a link fails immediately after transmitting a topology message. It does not occur when a single link or node fails. In most practical situations, the algorithm sends $2lk$ messages. The only serious concern is that this behavior could cause many algorithm messages to be queued for transmission on a link. However, this cannot occur since the most information that ever needs to be queued is the contents of a single port table. Any new status information simply replaces the old status, and does not add to the size of the queue.

The amount of processing required by the algorithm can be greatly reduced by constructing the main topology iteratively rather than by doing a shortest path calculation on each port topology table, as is implied by rule 5. A node first enters the status' of its adjacent links into its main table. It then finds all links one hop away and enters their status'. This continues for links two hops away etc. until the status of each connected link has been entered.

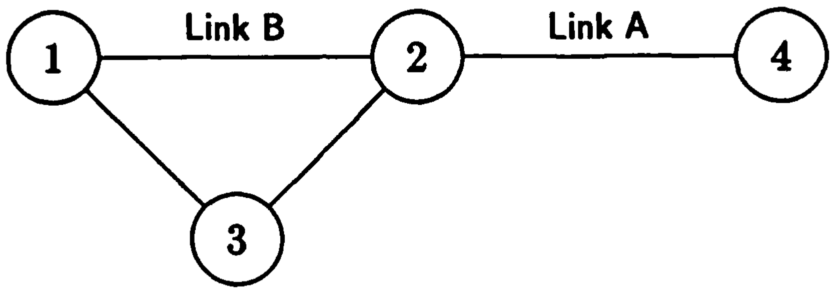

Figure 2a. **An Example Network.** All links are initially up. Link A fails. Shortly afterward, link B fails.

Event	Description
1	Link A fails. (2→1, $A\downarrow$) and (2→3, $A\downarrow$) sent and received.
2	(3→2, $A\downarrow$), (1→3, $A\downarrow$), and (1→2, $A\downarrow$) sent and received. (3→1, $A\downarrow$) sent but not yet received.
3	Link B fails. (1→3, $B\downarrow$), (1→3, $A\uparrow$), and (2→3, $B\downarrow$) sent and received.
4	(3→1, $A\downarrow$) that was sent in Event 2 arrives. (1→3, $A\downarrow$) sent and received.

Figure 2b. **Sequence of Events.** The table shows the messages sent in response to the change in status of links A and B. *Notation:* "($i \rightarrow j$, $l\downarrow$)" indicates a message sent from node i to node j saying that link l is down.

Event	T^1	T^1_2	T^1_3	T^2	T^2_1	T^2_3	T^2_4	T^3	T^3_1	T^3_2	T^4	T^4_2
1	du	du	uu	du	du	du	du	du	uu	du	du	du
2	du	du	uu	du	du	du	du	du	du	du	du	du
3	ud	dd	ud	dd	dd	dd	dd	dd	ud	dd	du	du
4	dd	dd	dd	dd	dd	dd	dd	dd	dd	dd	du	du

Figure 2c. **Topology Table Contents.** All topology table updating is assumed to take place immediately following an event. *Notation:* "du" means that in the indicated topology table, the entry for link A is down and the entry for link B is up.

Figure 2: SPTA Operation for Multiple Link Failures.

It is interesting to examine the behavior of SPTA when the assumption that nodes have error free memories is violated. Errors which occur in the main topology table are of no concern since they can always be corrected by using the port topologies and the known status of adjacent links. Similarly, *detected* errors in a port topology can be corrected by requesting retransmission from the appropriate neighbor. However, like all event driven algorithms, SPTA has no ability to correct *undetected* database errors. Without periodic retransmission an undetected error can persist for an arbitrary length of time. One obvious solution is using coding on the port tables to make the probability of an undetected error sufficiently small. In situations where extraordinary reliability is needed, a node could periodically retransmit its main table to each of its neighbors. This would destroy the event driven property of SPTA, but would maintain its other desirable characteristics, and allow recovery from undetected database errors.

4 Handling Routing Information

In order to make routing decisions, nodes often need a recent measure of the congestion in each direction on each network link. We are not concerned with how these measurements are made, but wish to design a distributed algorithm which will deliver them to each node in a connected set despite the presence of topological changes. This can be considered a generalization of the topology problem with directed links and nonbinary link status'.

A simple extension of SPTA can be used to solve this problem. A physical link l connecting nodes i and j is considered to be (for the purpose of congestion measurements) two directed links $i \rightarrow j$ and $j \rightarrow i$. A node is considered to be adjacent to its *outgoing* directed links, and at random times measures their congestion. Let C^i be a node i's main congestion table which contains a congestion measurement for each directed network link. Similarly, let C^i_j be node i's port congestion table associated with neighboring node j. To handle routing information, SPTA is extended to include the following rules.

1. When an entry in a node's main congestion table changes, a message containing the new value is sent on each operating adjacent link.

2. When a node receives a congestion update message from a neighbor which concerns a nonadjacent directed link, it enters the change in the port congestion table associated with that neighbor.

3. When a node detects that an adjacent link has become operational, it transmits its main congestion table on that link.

4. When an adjacent directed link changes its congestion value, the change is entered in the main congestion table.

5. When the entry for a directed link in one of a node's port congestion tables changes, it recomputes the main congestion table entry for that link by chosing the port table entry with the smallest distance measurement. A directed link has the same distance measurement as the corresponding undirected link.

6. When an entry in one of a node's port *topology* tables changes, it recalculates each entry in its main congestion table according to above rule 5.

Correctness of the above can be established by using an inductive argument which parallels the proof of SPTA. With this extension, SPTA can be used as a practical event driven alternative to standard algorithms such as the ARPANET update procedure.

Acknowledgements

The author thanks his research advisor, Dr. Robert G. Gallager, for his invaluable insight, and Dr. Dimitri P. Bertsekas for his help in improving the presentation of SPTA.

References

[1] E. C. Rosen, "The Updating Protocol of the ARPANET's New Routing Algorithm: A Case Study in Maintaining Identical Copies of a Changing Distributed Data Base," in *Proc. 4th Berkeley Conference on Distributed Data Management and Computer Networks*. Aug. 28-30 1979, pp. 260–274.

[2] J. M. McQuillan, I. Richer, and E. C. Rosen, "The New Routing Algorithm for the ARPANET," *IEEE Trans. Comm.*, vol. COM–28, No. 5, May 1980.

[3] A. Segall and M. Sidi, "A Failsafe Distributed Protocol for Minimum Delay Routing," *IEEE Trans. Comm.*, vol. COM–29, No. 5, May 1981.

[4] S. G. Finn, "Resynch Procedures and a Fail-Safe Network Protocol," *IEEE Trans. Comm.*, vol. COM–27, No. 6, June 1979.

[5] J. A. Roskind, "Edge Display Spanning Trees and Resynchronization in Data Communication Networks," Ph.D. Thesis, M.I.T., and M.I.T. Laboratory for Information and Decision Systems Report LIDS-TH-1332, Oct. 1983.

[6] A. Segall and B. Awerbuch, "A Reliable Broadcast Algorithm," M.I.T. Laboratory for Information and Decision Systems Report LIDS-P-1177.

[7] U. Vishkin, "An Efficient Distributed Orientation Algorithm," *IEEE Trans. Inform. Theory,* vol. IT-29, No. 4, July 1983.

[8] R. Perlman, "Fault-Tolerant Broadcast of Routing Information," *Computer Networks,* vol. 7, Dec. 1983.

[9] R. G. Gallager, "Distributed Minimum Hop Algorithms," M.I.T. Laboratory for Information and Decision Systems Report LIDS-P-1175, Jan. 1982.

[10] J. M. Spinelli, "Broadcasting Topology and Routing Information in Computer Networks," S.M. Thesis, M.I.T., and M.I.T. Laboratory for Information and Decision Systems Report LIDS-TH-1470, May 1985.

Distribution List

Defense Documentation Center Cameron Station Alexandria, Virginia 22314	12 Copies
Assistant Chief for Technology Office of Naval Research, Code 200 Arlington, Virginia 22217	1 Copy
Office of Naval Research Information Systems Program Code 437 Arlington, Virginia 22217	2 Copies
Office of Naval Research Branch Office, Boston 495 Summer Street Boston, Massachusetts 02210	1 Copy
Office of Naval Research Branch Office, Chicago 536 South Clark Street Chicago, Illinois 60605	1 Copy
Office of Naval Research Branch Office, Pasadena 1030 East Greet Street Pasadena, California 91106	1 Copy
Naval Research Laboratory Technical Information Division, Code 2627 Washington, D.C. 20375	6 Copies
Dr. A. L. Slafkosky Scientific Advisor Commandant of the Marine Corps (Code RD-1) Washington, D.C. 20380	1 Copy

Office of Naval Research — 1 Copy
Code 455
Arlington, Virginia 22217

Office of Naval Research — 1 Copy
Code 458
Arlington, Virginia 22217

Naval Electronics Laboratory Center — 1 Copy
Advanced Software Technology Division
Code 5200
San Diego, California 92152

Mr. E. H. Gleissner — 1 Copy
Naval Ship Research & Development Center
Computation and Mathematics Department
Bethesda, Maryland 20084

Captain Grace M. Hopper — 1 Copy
Naval Data Automation Command
Code OOH
Washington Navy Yard
Washington, DC 20374

Advanced Research Projects Agency — 1 Copy
Information Processing Techniques
1400 Wilson Boulevard
Arlington, Virginia 22209

Dr. Stuart L. Brodsky — 1 Copy
Office of Naval Research
Code 432
Arlington, Virginia 22217

Prof. Fouad A. Tobagi
Computer Systems Laboratory
Stanford Electronics Laboratories
Department of Electrical Engineering
Stanford University
Stanford, CA 94305